AC 2010-1431: INTEGRATION OF GRAPHICAL PROGRAMMING INTO A FIRST-YEAR ENGINEERING COURSE

Gregory Bucks, Purdue University

GREGORY W. BUCKS is a PhD candidate in the school of Engineering Education at Purdue University. He received his BSEE from the Pennsylvania State University and his MSECE from Purdue University. His research interests lie in the development of conceptual understanding of computer programming concepts and the exploration of the pedagogical benefits of graphical programming languages.

William Oakes, Purdue University

William Oakes is the Director of the EPICS Program and an Associate Professor and a founding faculty member of the Department of Engineering Education at Purdue University with courtesy appointments in Mechanical Engineering and of Curriculum and Instruction. He is a co-recipient the NEA's Bernard M. Gordon Prize for Innovation in Engineering and Technology Education, the Campus Compact Thomas Ehrlich Faculty Award for Service-Learning; the NSPE's Educational Excellence Award.

Integration of Graphical Programming into a First-Year Engineering Course

Abstract

Many first-year engineering curricula either include a course on computing or integrate computing within one of the introductory courses. There is significant evidence that students in these introductory programming courses have difficulty both learning the concepts as well as applying those concepts in the writing of code. This is especially prominent among engineering students. One reason for this discrepancy between the learning outcomes desired by instructors and student performance is that the instructional methods used and the very nature of the material does not match well with the learning styles of most engineering students. One promising avenue to explore in an attempt to address this issue is the use of graphical programming environments. A hypothesis is that using environments such as these could help students who tend to learn better from visual presentations, because the graphical nature of the program will help to make the structures easier to comprehend. The potential to enhance first-year student learning motivated a pilot approach at a large university's introductory engineering class to use graphical programming as the dominant computer tool within the class. Design was taught through the use of a graphical programming language that culminated in a service-learning project in which students developed computer programs designed to excite middle school students about math and science. This paper will discuss the curricular structure, the implementation of the graphical programming language, examples from the class and initial assessments from the experience.

Introduction

Computers are an integral part of the engineering landscape. They are used to model potential solutions, collect and analyze data, and create new parts through computer aided design packages and computer controlled machinery. In addition, they are starting to be increasingly incorporated into the products of design, from sneakers that track the distance traveled to smart building materials that can report on the stresses and strains they are experiencing. Computing skills have been identified as one of the attributes that future engineers will be required to possess [1]. Due to this increasing prevalence of computing technologies in both the design and implementation of engineering solutions, many first-year engineering curricula include either a course devoted entirely to computing concepts, or incorporate those concepts into other introductory courses.

There is significant evidence that students in introductory programming courses have difficulty both learning the concepts inherent in the field or computer science as well as applying those concepts in the writing of code [2, 3]. This is also true of engineering students. One reason for this discrepancy between the learning outcomes desired by instructors and student performance is that the instructional methods used as well as the very nature of the material do not match well with the learning styles of most engineering students.

There are many learning style models and assessments that have been developed. The most commonly used within engineering is the Felder-Silverman learning styles model [4], with its associated assessment, the Index of Learning Styles (ILS). This model categorizes students based on four dimensions, which characterize different aspects of student learning: sensing versus intuitive, visual versus verbal, active versus reflective, and sequential versus global. In terms of learning programming content, the two most important scales are the visual versus verbal scale and the sensing versus intuitive scale.

There have been numerous studies that have looked at the learning styles preferences of engineering students [5-7], and those preferences are consistent across populations [8]. What these studies have found is that engineering students tend to be more visual and sensing in their learning styles preferences. However, since most programming languages taught in introductory courses are text-based, there is a mismatch between what is being taught and how many students prefer to learn. It has been shown that interpretation of the written word, while presented in a visual manner, is processed in the same way as spoken words [9]. This means that students who prefer to learn in a visual manner may have difficulty assimilating programming content generally conveyed in a verbal context. The sensing versus intuitive scale is also a contributor to this problem, in that interpreting the meaning of words favor students who favor using their intuition. This again causes issues for most engineering students, who tend more toward the sensing end of the scale.

In addition, many students lack appropriate models on which to build conceptions of the important programming structures they are required to learn [10]. In conjunction with this, many text-based languages use syntax that incorporates many English terms, causing students to resort to using the models they have developed for the natural language use of these terms. However, this poses a significant problem for some terms because the model for how the word is used in natural language differs from how it is used in a programming context.

For example, in natural language, the term “while” has a slightly different meaning than it does in programming usage. In natural language, “while” implies that as soon as whatever the condition tied to the statement is no longer satisfied, the activity will cease. In a programming context, the conditional statement associated with the “while” is only checked once during an iteration. This can cause students issues if they believe that as soon as their condition is met, the loop will exit.

One possibility for addressing these issues is through the use of graphical programming languages. Graphical languages can provide a way to introduce programming concepts in a way that caters to individuals who tend toward more visual styles of learning. In addition, it may also provide a medium for students to begin to develop their own models of programming concepts, as the visual nature can provide a structure for organizing their models and, for the most part, the graphical representation is free from the natural language terminology used in most text-based languages. This relative independence from natural language terminology may help lead students away from resorting to their incorrect natural language models and force them to develop more complete models of the programming concepts themselves.

Another benefit of utilizing graphical languages as the medium through which engineers learn computer programming is the ability to incorporate elements of engineering design. In addition to computing, design is an important concept for engineers and engineering students. Challenges exist for educators trying to introduce students to design early in their academic careers. One challenge is that students do not have much knowledge upon which to build a design from. They have not had their engineering coursework yet and do not have the tools to do sophisticated designs. A fall back is to have students do simple designs that do not require much, if any, iteration and hardly any analysis. Students can have fun working on these design projects, but they, in general, are not real designs and the students know they are not real. Trying to introduce a human-centered design approach is doubly challenging for early students because they are limited to what they can actually do. There therefore exists an opportunity if students can be quickly equipped with skills to create a "real" design for real users. Graphical programming is such a tool that also addresses the issues in computing. This paper will lay out how graphical programming was used in first-semester design class.

Curricular Structure

The first-year engineering program at Purdue University is a required two-semester sequence for all engineering majors. The class is broken up into sections of 120 students per section. The first-semester course goals and course objectives are listed below.

Course Goals

- Link own career goals with nature of engineering and the characteristics of various engineering disciplines,
- Use modeling tools to make design decisions,
- Use a design process to solve complex engineering design challenges.

Course Objectives

- Examine and analyze career information from various resources to make informed decisions about which engineering discipline to pursue,
- Explain the critical role of cross-cultural and multidisciplinary teamwork in nurturing diverse perspectives and the creation of innovative engineering solutions that meets the needs of diverse users,
- Develop metacognitive skills in evaluating own teamwork and leadership abilities, recognizing how own behavior impact the whole team, and make team process adjustments when necessary,
- Explain critical and diverse use of modeling in engineering to understand problems, represent solutions, compare alternatives, make predications, etc,
- Use multiple models, estimation, and logic to triangulate and evaluate information coming from various data sources,

- Collect, analyze, and represent data to make informative explanations and persuasive arguments,
- Implement iterative processes, rich information gathering, and multiple modes of modeling when solving complex design problems,
- Use systematic methods to develop design solutions and compare design alternatives, and
- Consider the interconnectedness among social, economic, environmental factors (in the context of sustainability or systems) when solving engineering problems.

For the fall 2009 semester, two sections of the course were modified to incorporate the use of a graphical programming environment to explore its use for both teaching introductory computing and design concepts. The graphical language implemented was National Instruments LabVIEW. LabVIEW was chosen due to the familiarity of the instructors and staff as well as its availability on campus.

LabVIEW is a graphical programming language in which an individual creates a program by connecting different graphical blocks together, similar to a circuit diagram or block diagram. The programmer creates both the user interface for the program as well as the code simultaneously. The user interface is created using the Front Panel window, on which different objects, such as numeric inputs and outputs, graphs, and text displays are placed to allow a user to provide inputs to and receive outputs from the program. Objects placed on the Front Panel are automatically linked to the Block Diagram, which is where the code is created. The code, as stated above, is represented as a block diagram, where different functions are depicted using functional blocks and connected together using lines to specify where the data should flow within the program. An example Front Panel and Block Diagram are shown below.

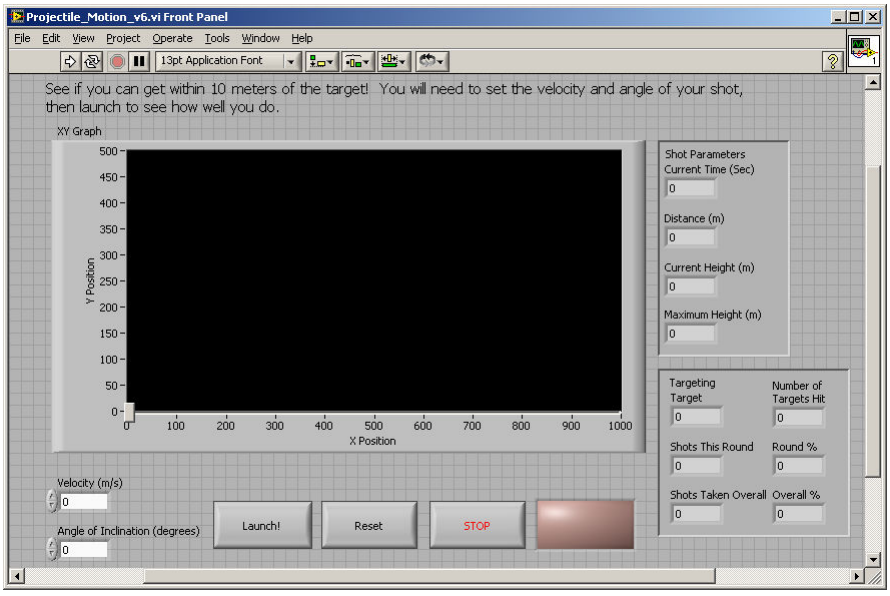

Figure 1: Example of a LabVIEW Front Panel

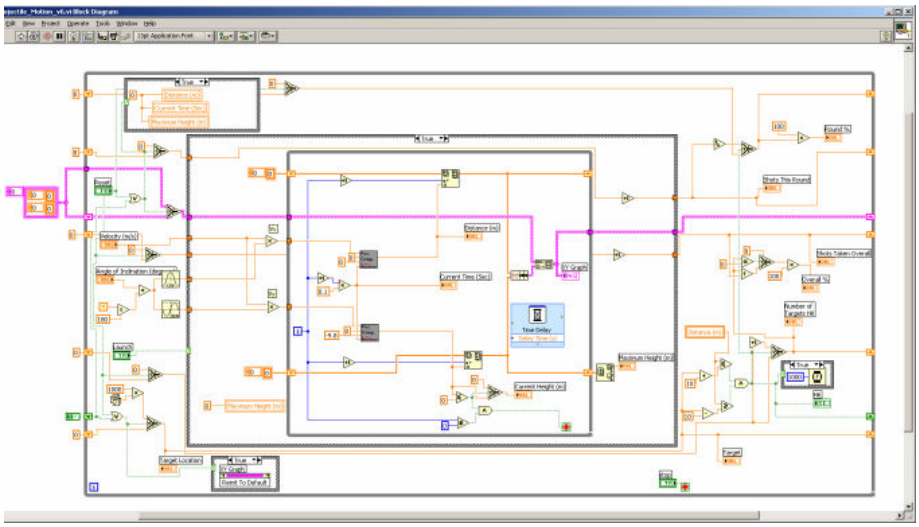

Figure 2: Example of a LabVIEW Block Diagram

The computing concepts covered in LabVIEW were first introduced in the lecture period and reinforced through activities in the laboratory period. Overall, it took 6 lecture and lab periods to introduce students to all of the fundamental concepts covered in the course.

In order to tie the design and computing aspects of the course together, a service-learning project was presented in the middle of the semester. Students were required to develop a program in LabVIEW that would teach a concept from the Indiana State math or science curriculum for the 7th grade. Students were provided with the Indiana State standards for math and science and were allowed to choose the concept they would implement. Near the conclusion of the project, the students presented their projects to several 7th grade classes from the local community. Following this, the students were required to propose updates to their programs based on the feedback they received from the 7th grade students. Throughout the design and development of their programs, students had access to several LabVIEW experts who provided additional instruction on the capabilities of the LabVIEW language not covered in the class. Examples of what the students were able to create are discussed in the next section.

As discussed above, another topic that was covered in the course was what each of the different engineering majors do. In the other sections of the course, this was accomplished through the development of a board game. In the two experimental sections, the board game implementation was replaced with the development of a LabVIEW program that would ask a series of questions and provide results showing what engineering major might be the best fit based on the responses.

Examples of Student Work

Several examples of the programs that the students were able to create are shown below. To download a file with all of the student projects, go to the following link: https://engineering.purdue.edu/EPICS/ENGR195_Project/Fall2009_ENGR195

This project was developed by a group of students intending to teach basic math concepts such as multiplication and division. The students wrapped these concepts in a flight simulator, where

every question answered correctly causes the plane to increase its altitude and every incorrect answer causes the plane to drop in altitude. The objective is to try and reach a safe cruising altitude. The students imported a picture of a instrument panel for a plane onto the Front Panel and placed several LabVIEW objects over top of the original image to make the simulation dynamic.

Figure 3: Math Flight Simulator

Another project that aimed to teach basic math concepts used the idea of popular game from the late 1980's and early 1990's as a template. They created a game similar to the educational Oregon Trail video game produced by MECC, in which students advance along the Oregon Trail by answering math questions correctly. Students are allowed to attempt a difficult problem first. If they answer the question correctly, they move on to the next position and 2 days are added to their total days on the trail. If they answer the question incorrectly, they receive a 3 day penalty and attempt to answer the question again. If they miss the question again, they receive another penalty, but are able to attempt an easier problem. There are total of thirteen stops, with hard and easy questions for each. In addition to simply asking the questions and keeping track of the number of days on the trail, the students were also able to incorporate a series of pictures that change based on the location on the trail and a sound file that plays at the beginning and end of the program.

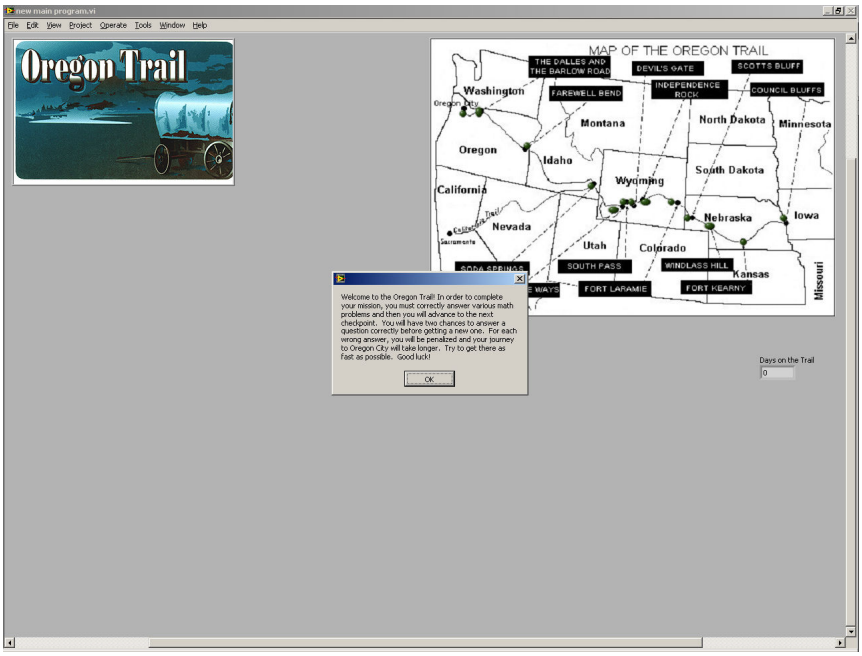

Figure 4: Oregon Trail

The next project was designed to teach students about both projectile motion and gravity on different planets. The objective is to try and launch the bowling ball over the brick wall and hit the bowling pins by changing the angle and velocity at which the bowling ball is shot out of the cannon. The user starts on Earth and progresses through all of the planets in the solar system, moving to the next planet once the pins have been successfully hit. A picture of the current planet appears at the top of the screen, and the acceleration due to gravity is displayed in the window at the top as well.

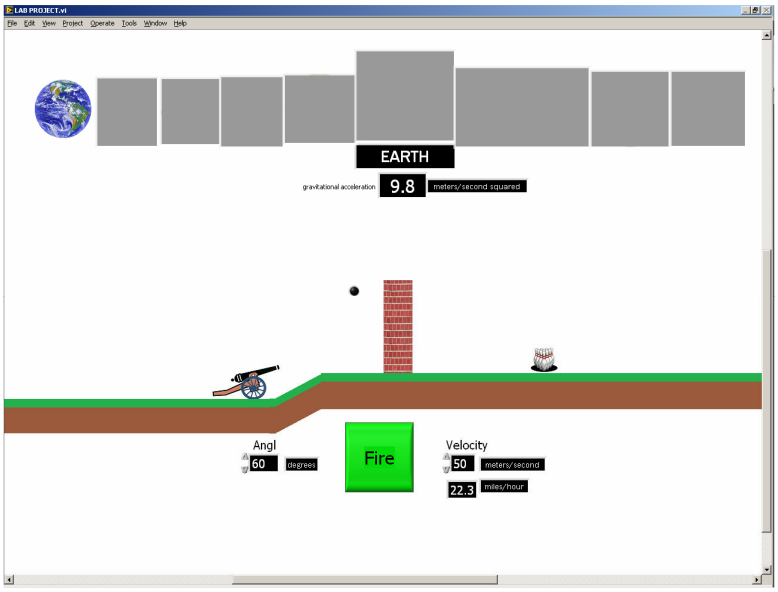

Figure 5: Projectile Motion on Different Planets

This last project teaches students about different concepts related to waves, such as frequency and wavelength. There are two aspects to this project. First, on a computer equipped with a microphone, a student can record his/her voice then play it back at different speeds by changing the slider on the left. The original and modified waveforms are displayed on the graphs in the middle of the window. In addition, students can take a quiz about the different concepts related to waves.

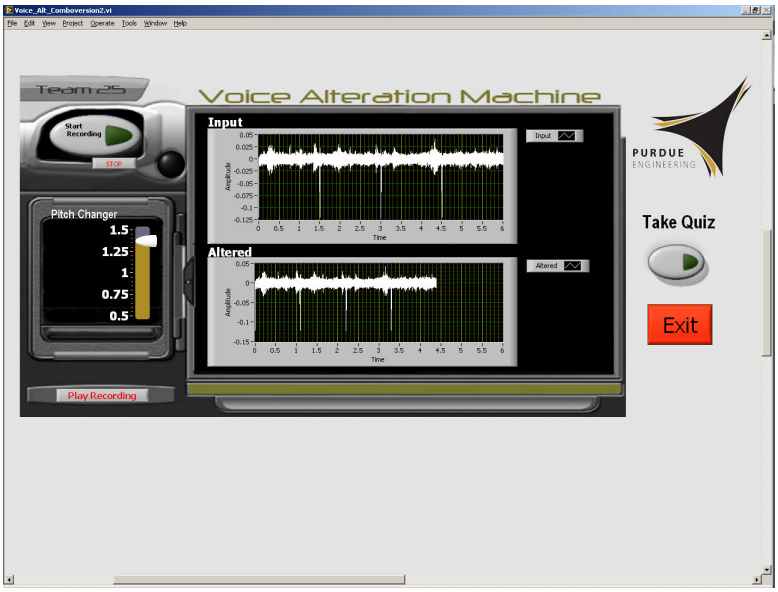

Figure 6: Voice Modification Program

Participants

The students in the two experimental sections came from two different sources. One of the sections consisted of students who were in two of the engineering learning communities at Purdue University. Learning communities provides a student the opportunity to live together with other students who have similar interests and take several introductory classes together. This can aid in the transition to college by providing a smaller community within the university. One of the learning communities involved participation in a service-learning design course, so these students knew they would be working on projects throughout the semester, but did not know prior to the beginning of the semester that their section of the introductory course would be different than other sections.

The second experimental section of the course was a more traditional section in terms of the student population. Students were randomly assigned to the section out of the total first-year engineering population. They also did not know this section would be any different than the others going into the semester.

Of the 150 students who participated in the course and completed the end-of-semester survey (228 total students participated in the course, 65% response rate), 62 had some type of prior programming experience (41%) and 88 had no prior experience (59%). Of those students who had prior programming experience, 76% reported that some of their experience came from courses in elementary, middle or high school, 24% from extracurricular activities, 22% from

self-taught experiences, 11% from work experiences, and 5% from a variety of other experiences.

In addition to reporting on their experiences, the students also reported on what languages they knew. Of the 62 students with prior programming experience, 44% had experience with Basic, 34% with Java, 29% with HTML, 21% with C++, 20% with C, 13% with Matlab, and 18% had experience with other languages.

Integrating 7th graders

Integrating the 7th grade students into the experience added some complexity but also a lot of value for all involved. The classes piloted were back-to-back so we only used one school. A call and then follow-up email to the principal got it all rolling. The follow-up email described our project and when we would need the students. The principal identified two math teachers who recruited the students, which made it very easy for us. Students were given a very brief overview when they arrived on campus about the project and what their role would be. They took to the role of evaluator very well and treated it very seriously. The evaluation form was a short set of Likert scale questions and the students answered them in teams. Teams ranged from 2 to 4. The larger teams were preferred but to provide enough teams to engaged all the undergraduates, we reduced the size to 2.

Preliminary Results

Initially, many of the students in the experimental sections had concerns about the additional workload and content to be covered, as it was significantly more than was required in the other sections of the course. However, as the semester progressed, especially once they began work on their projects and saw what they were capable of doing, they became quite excited. Many of the project teams went far beyond what was required of them for the projects, learning about different capabilities of LabVIEW that were not covered in class.

One group in particular (the Oregon Trail team from above) spent a great deal of time working on their project. The group consisted of four female students. They had the basic functionality of asking the questions and keeping track of the number of days on the trail working within the first week of the project. From there, they decided they wanted to add some additional features to make the game more exciting, such as showing the position along the trail on the map, playing music, and displaying different pictures at each of the stops along the trail. They were really excited about their project, and continued to make improvements and add additional features up until the day they presented the project to the 7th grade students. This type of reaction from female students toward computing is very different from the normal reaction of most female students toward computing. It shows how incorporating service-learning type experiences, where students can see the impact of their work on others, draws in and excites populations traditionally adverse to specific fields and disciplines.

These results are supported by an end-of-semester survey that was administered to the students in the experimental sections which asked about their attitudes toward programming in general and

about their experiences this semester with LabVIEW and the projects. As can be seen in the table below, the students responded positively to using LabVIEW and very positively to the projects. Surprisingly, despite all of their misgivings at the beginning of the semester, the students did not feel that the projects were too difficult. They also felt very positively about presenting their projects to the 7th grade students. The survey utilized a 5-point Likert scale, with 1 corresponding to strongly disagree and 5 corresponding to strongly agree.

LabVIEW Questions	Avg.	Project Questions	Avg.
I feel that LabVIEW was easy to use	3.23	I feel that the semester projects helped me understand programming better	3.82
I feel that being able to see the programming concepts in LabVIEW helped me to understand them	3.67	I feel that the semester projects were too difficult	2.74
I feel that LabVIEW Block Diagrams are confusing	2.72	I feel that I had enough help from the instructors and TAs to complete the semester projects	3.36
I feel that I have no trouble finding the functions that I want to use	3.15	I feel less confident in my programming ability after working on the projects	2.33
I feel that learning LabVIEW will help me to learn another programming language next semester	3.58	I feel that the semester projects helped me to see how engineering can have an impact	3.57
I feel that I can easily understand what is going on in a program written in LabVIEW by looking at the Block Diagram	3.50	I enjoyed working on the semester projects	3.42
I feel that the palettes are confusing	2.77	I feel that presenting our projects to the 7th grade students was a good experience	4.07
I feel comfortable using LabVIEW to create programs	3.42	I feel that working on the projects improved my confidence in my programming ability	3.71
I feel that being able to see the program visually helps me understand how the program works	3.75		

Table 2: Results of the end-of-semester survey

Overall course ratings were compared to the traditional course. There were twelve sections of the course and two were taught using the modified approach using the graphical programming.

The average course evaluations for the modified sections was almost one point higher on a scale of 1 to 5. This is striking since it was well known (and explicitly states in the course) that these sections were doing more work in the course and as homework than the traditional courses. The students seemed to appreciate the experience and what they had learned.

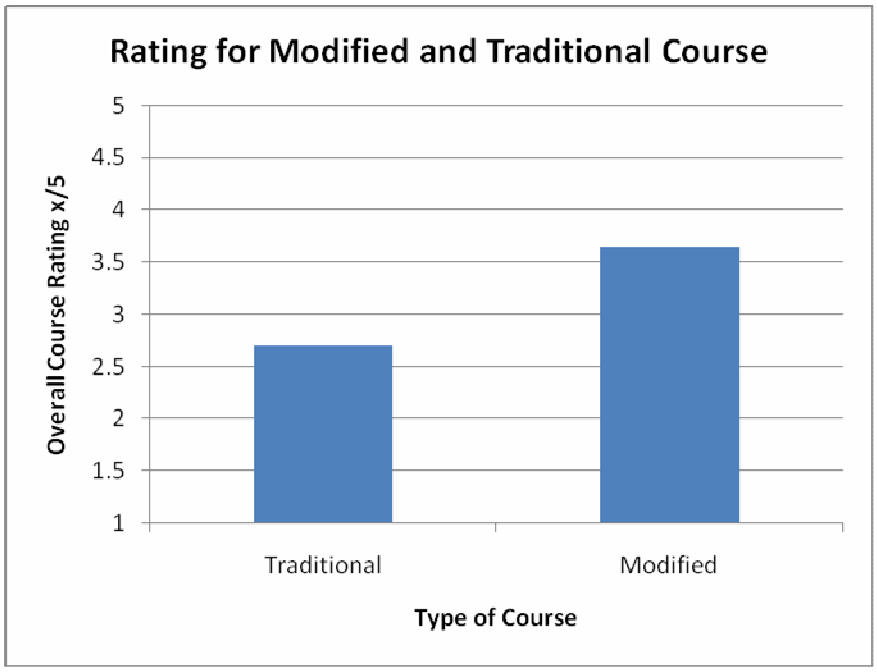

Figure 7: Overall Course Ratings

Conclusion

Based on preliminary analysis of the data collected and impressions from the instructional staff, the incorporation of LabVIEW into this introductory course was a success. The students demonstrated that they understood the programming material through their performance on the two projects and on their exams, which included programming concepts and were genuinely excited about what they were able to produce. They were able to create programs that were quite sophisticated, especially for first-year students, with only a third of a semester's worth of instruction. In addition, they were highly motivated by the semester projects, and for the most part, far exceeded the expectations of the instructors.

Industry professionals that were brought in to advise students, echoed the instructional team's assessment that the students really seem to enjoy and understand what they were doing. First-semester students were able to create programs that used LabVIEW in some innovative and creative ways.

The integration of the 7th graders was a key part of the experience. The 7th grade teachers reported that their students really loved the experience and left excited. The undergraduates reported that having real people look at their projects made it real to them and worthwhile.

The data also shows that the service-learning dimension of delivering a product to real middle school students was a motivator for the students. It pushed them to be more creative and also gave them a sense of satisfaction. The real users, the 7th graders, also allowed the concepts of user-centered design to be learned and appreciated.

Data will continue to be collected during the spring 2010 semester in order to compare how the performance and attitudes of students in the experimental section with students in the traditional sections, who had no programming experience during the fall 2009 semester and who will be learning Matlab and completing the same service-learning project. In addition, the students from the fall 2009 experimental section will also be learning Matlab, allowing study of how well the students are able to transfer their knowledge from graphical to text-based environments.

Bibliography

1. National Academy of Engineering, *The engineer of 2020 : visions of engineering in the new century*. 2004, Washington, DC: National Academies Press. xv, 101 p.
2. McCracken, M., et al., *A multi-national, multi-institutional study of assessment of programming skills of first-year CS students.* SIGCSE Bulletin, 2001. **33**(4): p. 125-180.
3. Thomas, L., et al., *Learning styles and performance in the introductory programming sequence.* SIGCSE Bulletin, 2002. **34**(1): p. p. 33-37.
4. Felder, R.M. and L.K. Silverman, *Learning and teaching styles in engineering education.* Engineering Education, 1988. **78**(7): p. 674-681.
5. Felder, R.M. and J. Spurlin, *Applications, reliability and validity of the index of learning styles.* International Journal of Engineering Education, 2005. **21**(1): p. 103-12.
6. Rosati, P.A. *The learning preferences of engineering students from two perspectives*. in *Frontiers in Education*. 1998. Tempe, Arizona.
7. Litzinger, T.A., et al., *A psychometric study of the index of learning styles.* Journal of Engineering Education, 2007. **96**(4): p. 309-319.
8. Zualkernan, I.A., J. Allert, and G.Z. Qadah, *Learning styles of computer programming students: a middle eastern and American comparison.* IEEE Transactions on Education, 2006. **49**(4): p. 443-50.
9. Felder, R.M., *Learning and teaching styles in foreign and second language education.* Foreign Language Annals, 1995. **28**(1): p. 21-31.
10. Bonar, J. and E. Soloway. *Uncovering principles of novice programming*. in *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 1983. Austin, Texas: ACM.